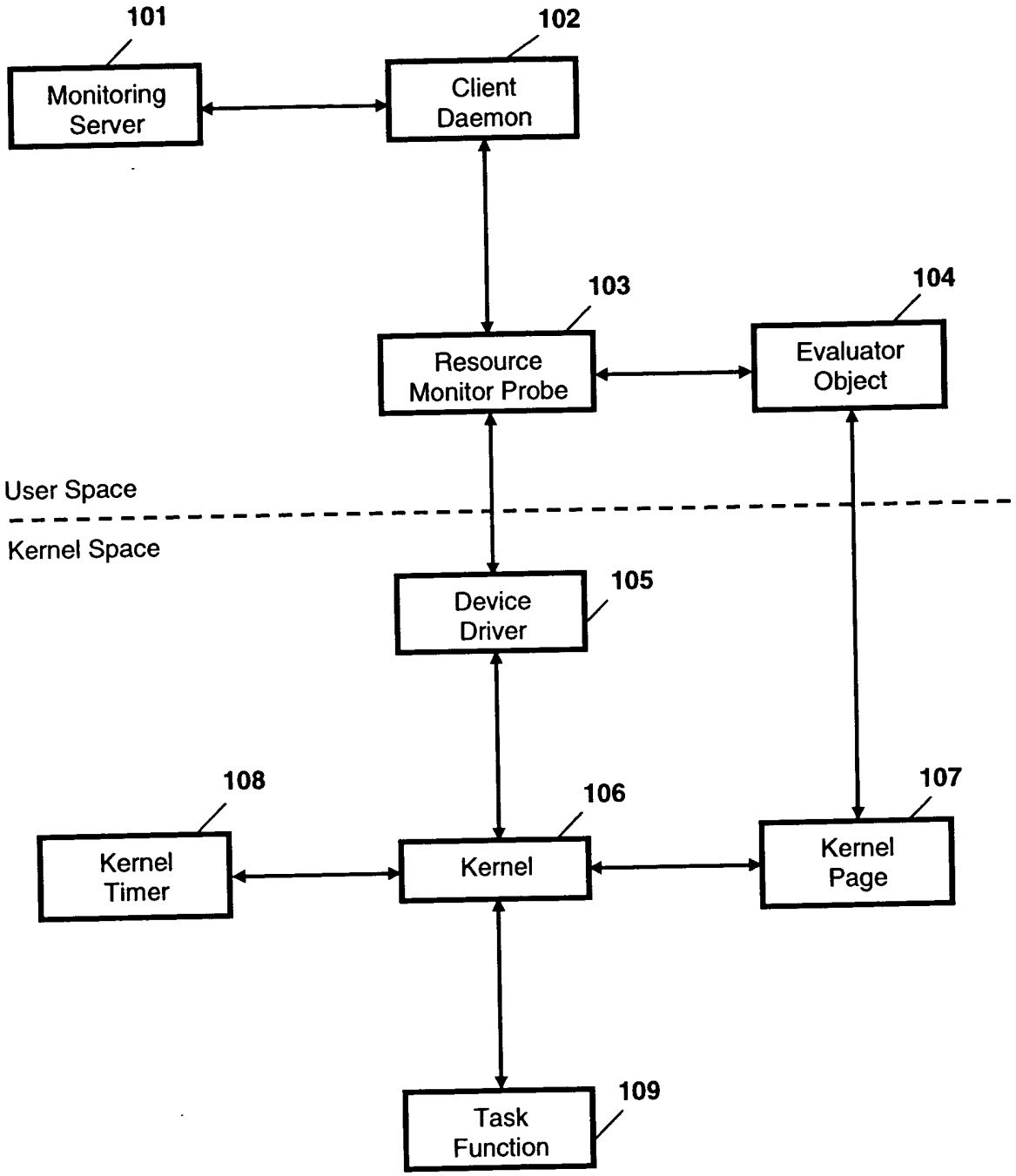

FIG. 1

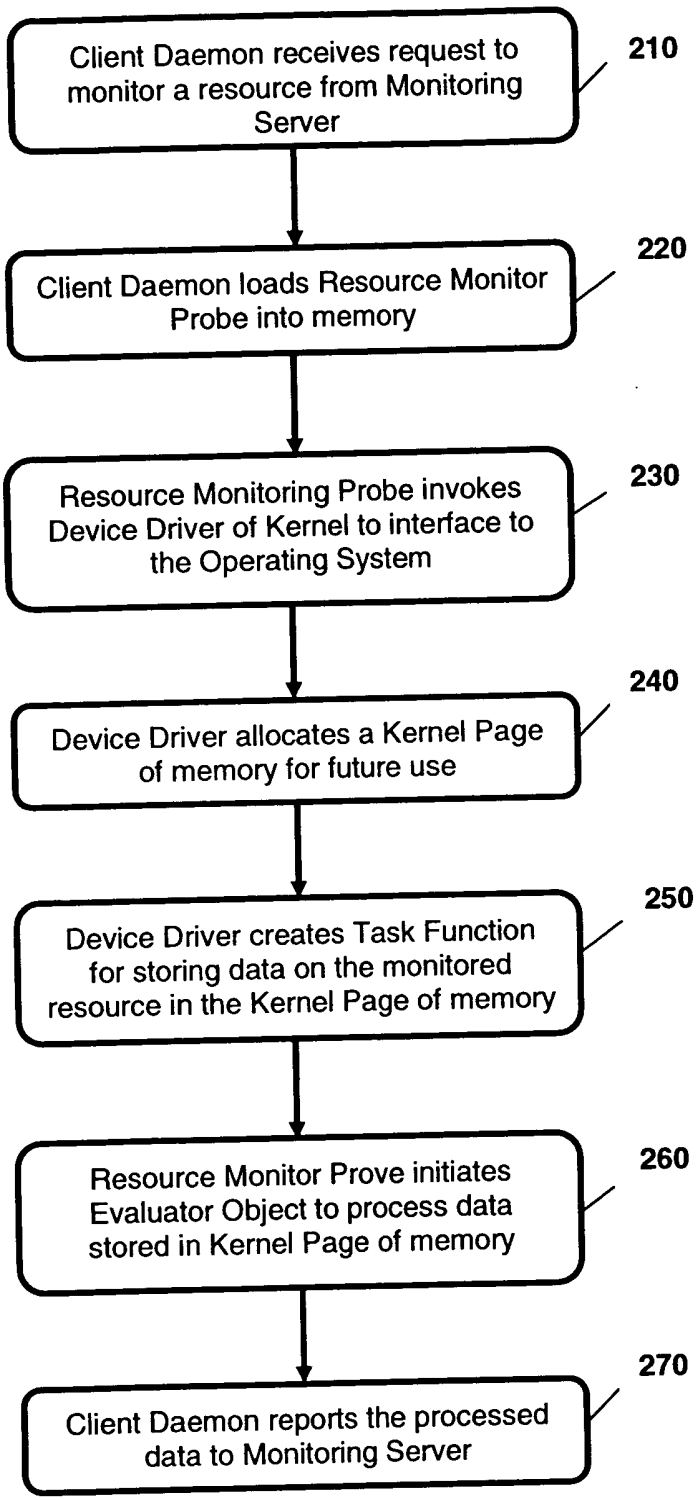
Client Daemon receives request to monitor a resource from Monitoring Server
210
Client Daemon loads Resource Monitor Probe into memory
220
Resource Monitoring Probe invokes Device Driver of Kernel to interface to the Operating System
230
Device Driver allocates a Kernel Page of memory for future use
240
Device Driver creates Task Function for storing data on the monitored resource in the Kernel Page of memory
250
Resource Monitor Prove initiates Evaluator Object to process data stored in Kernel Page of memory
260
Client Daemon reports the processed data to Monitoring Server
270

FIG. 2

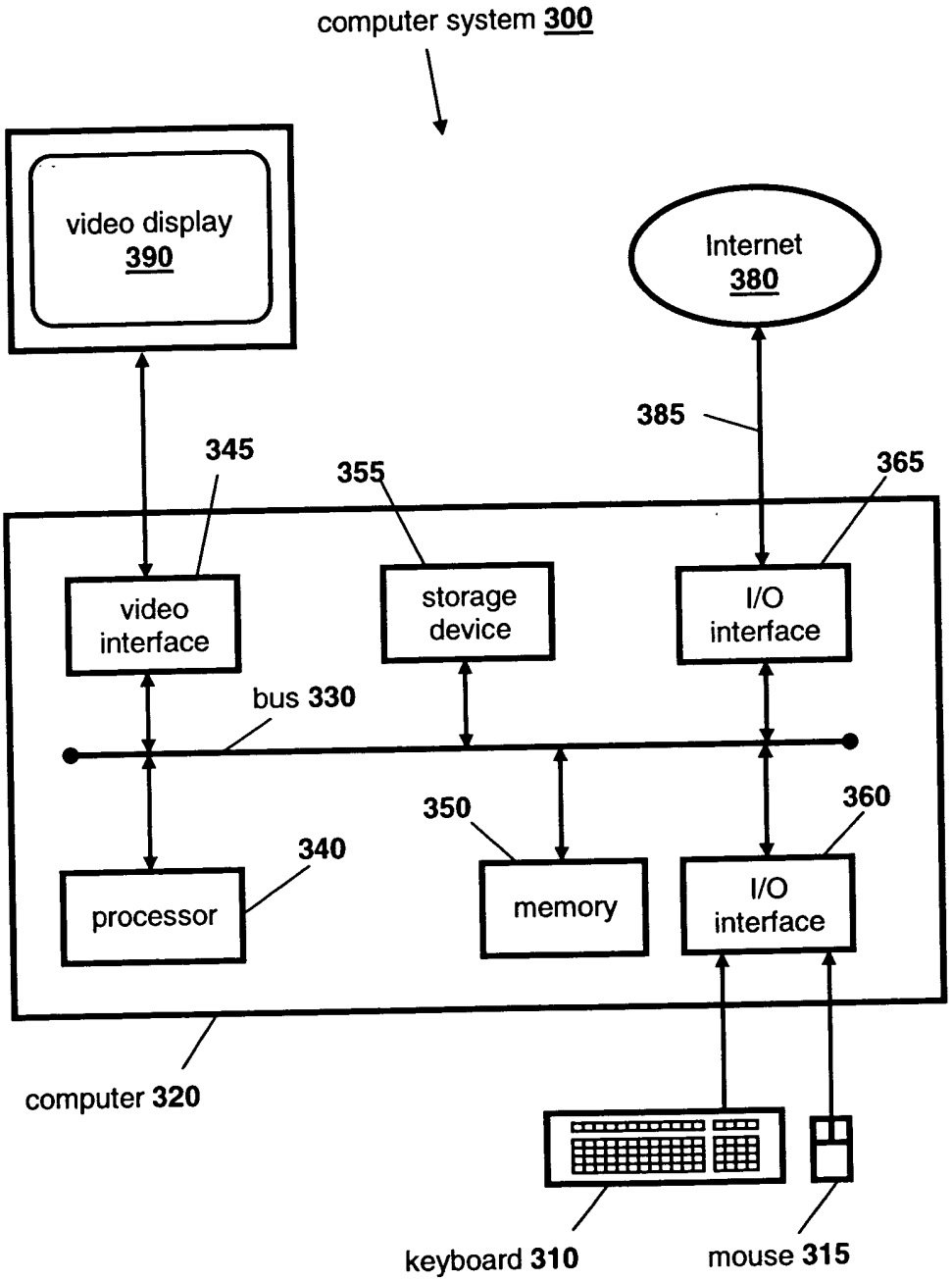
computer system 300
video display 390
Internet 380
385
345
355
365
video interface
storage device
I/O interface
bus 330
350
360
340
processor
memory
I/O interface
computer 320
keyboard 310
mouse 315

FIG. 3

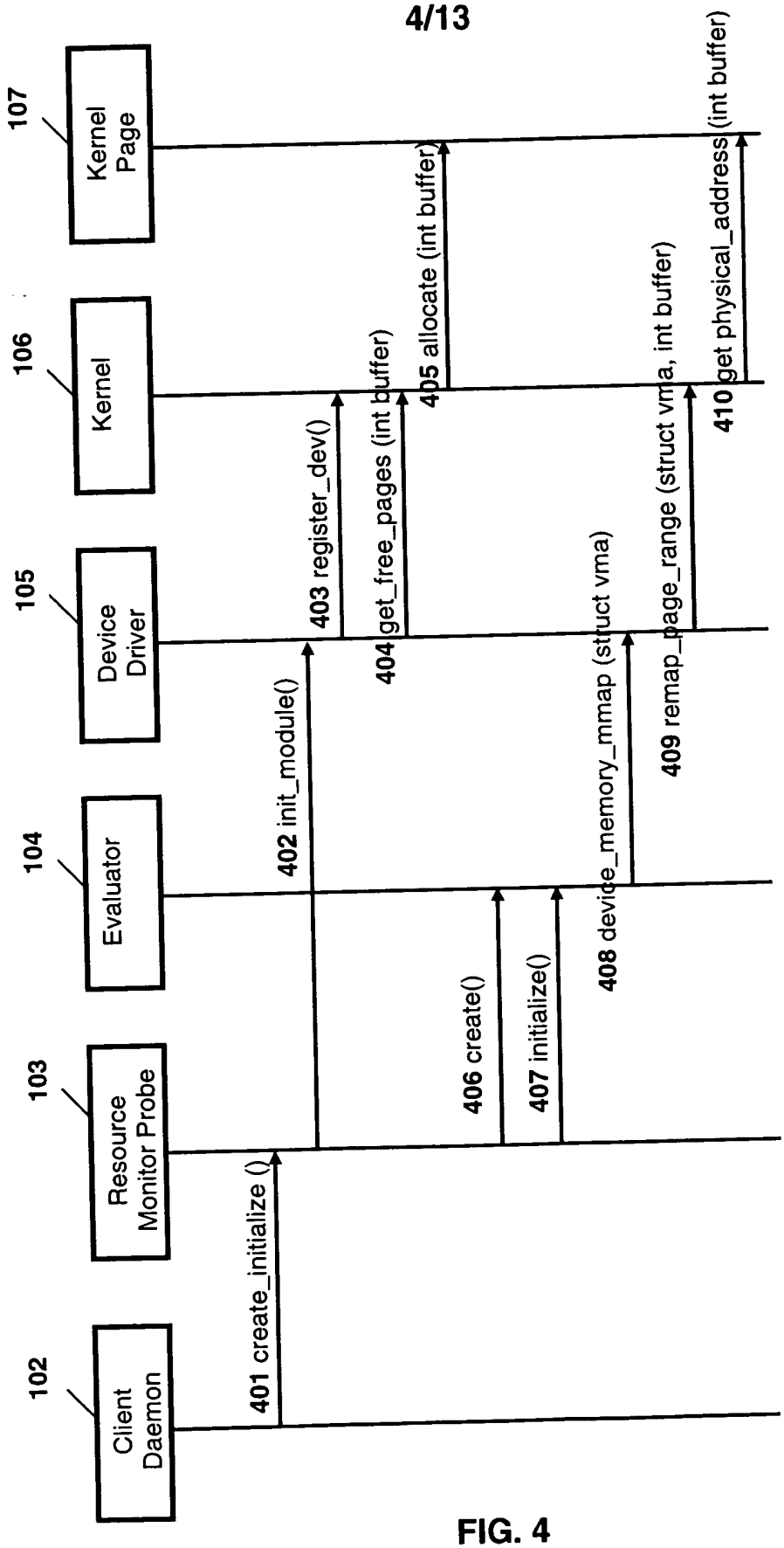

FIG. 4

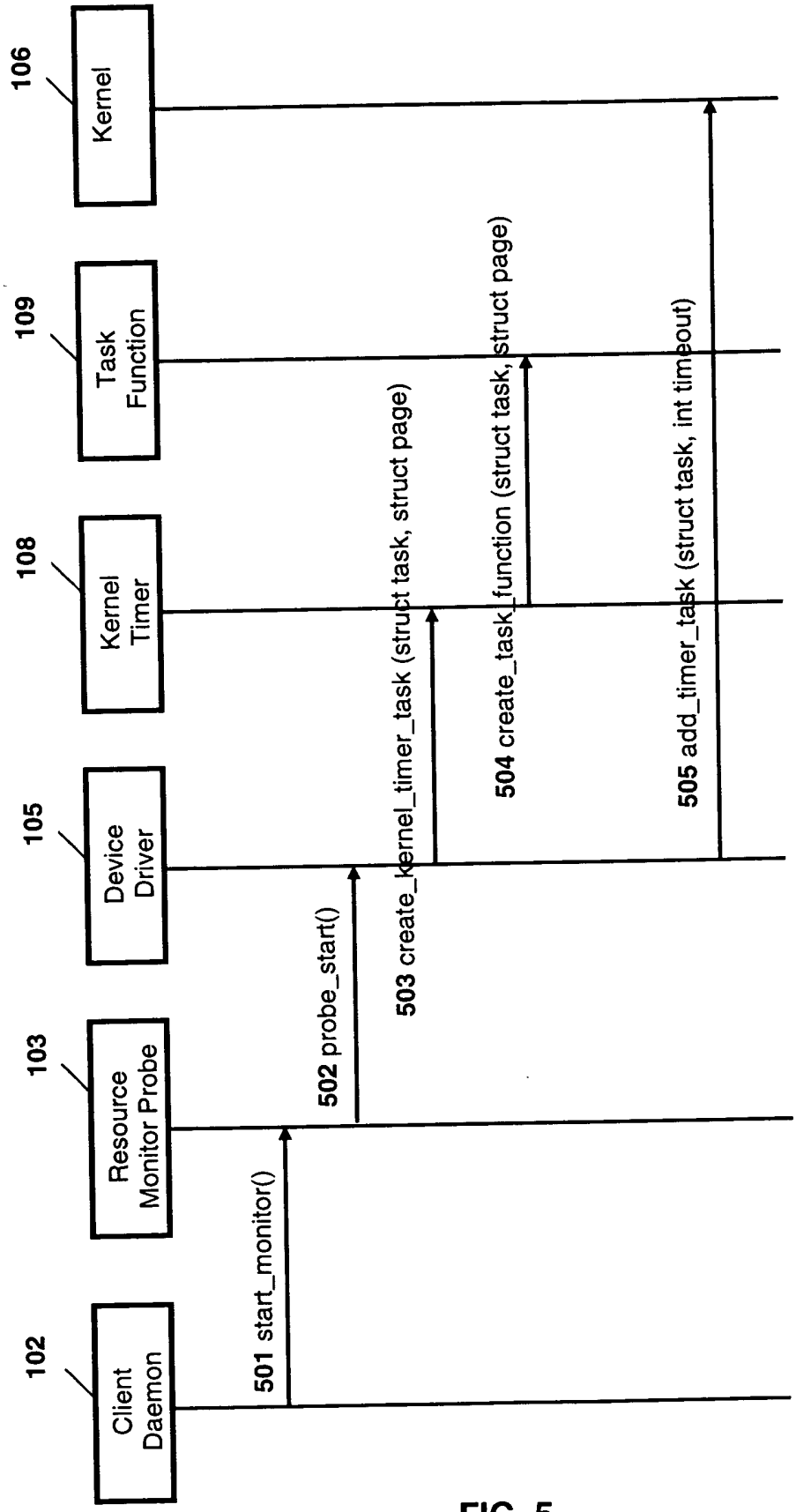

FIG. 5

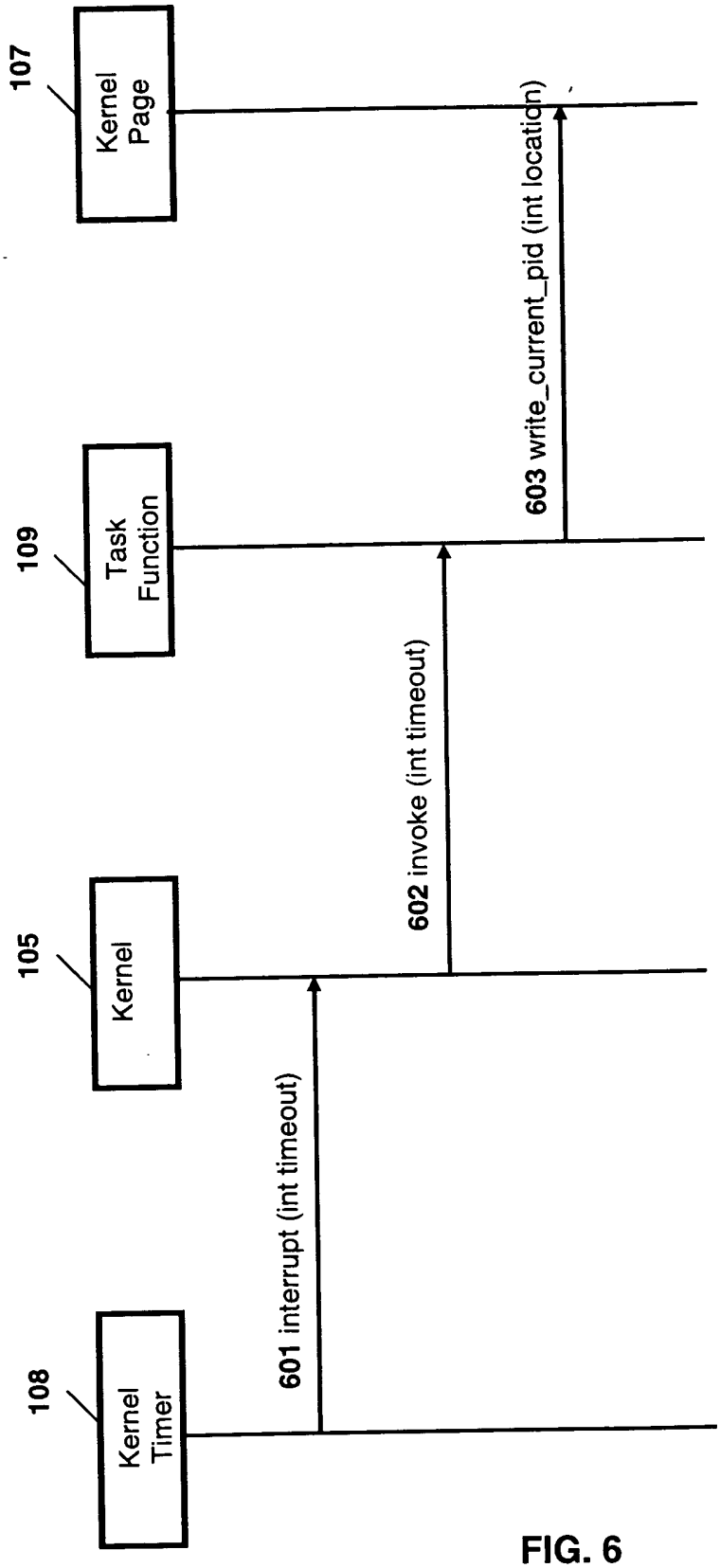
107
Kernel Page
109
Task Function
105
Kernel
108
Kernel Timer
603 write_current_pid (int location)
602 invoke (int timeout)
601 interrupt (int timeout)

FIG. 6

7/13

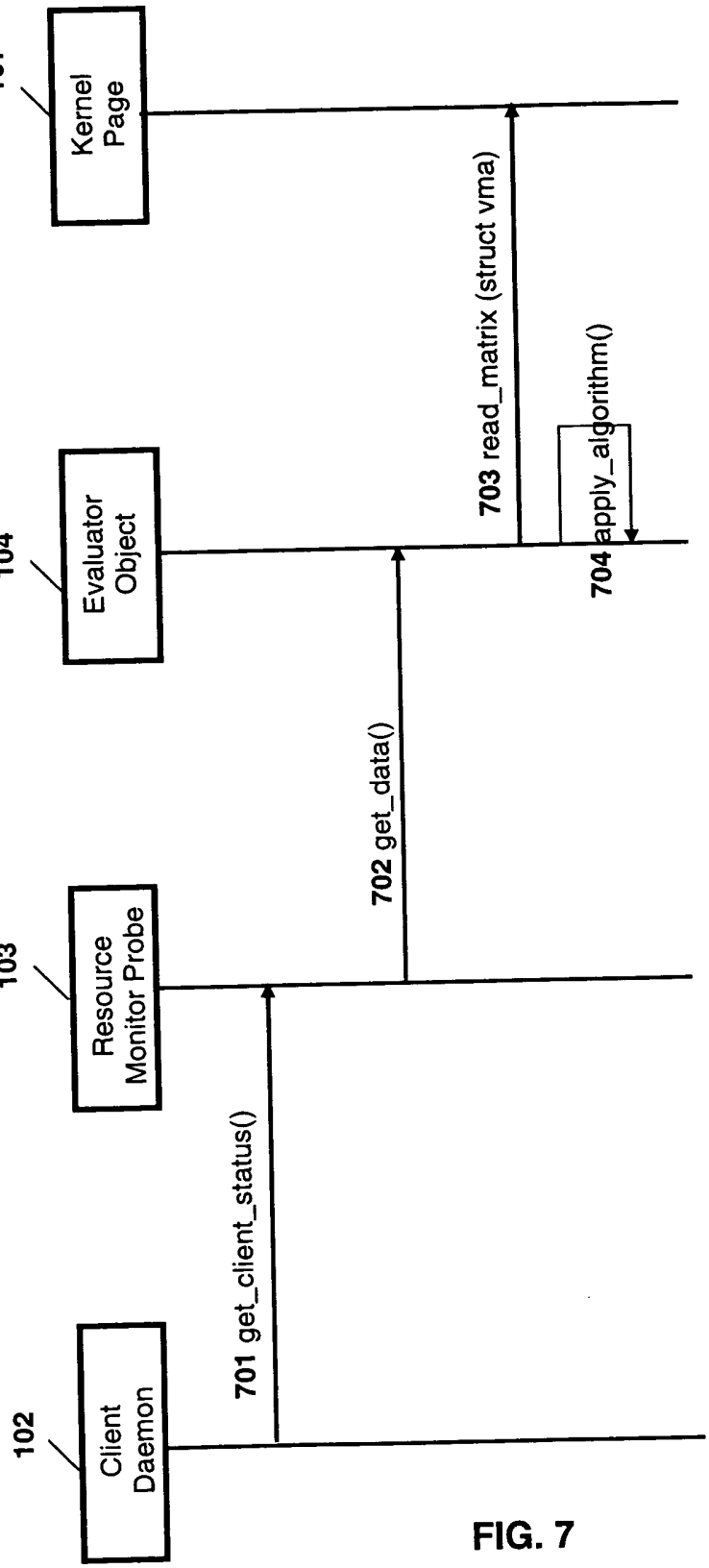
107
Kernel Page
104
Evaluator Object
103
Resource Monitor Probe
102
Client Daemon
703 read_matrix (struct vma)
704 apply_algorithm()
702 get_data()
701 get_client_status()

FIG. 7

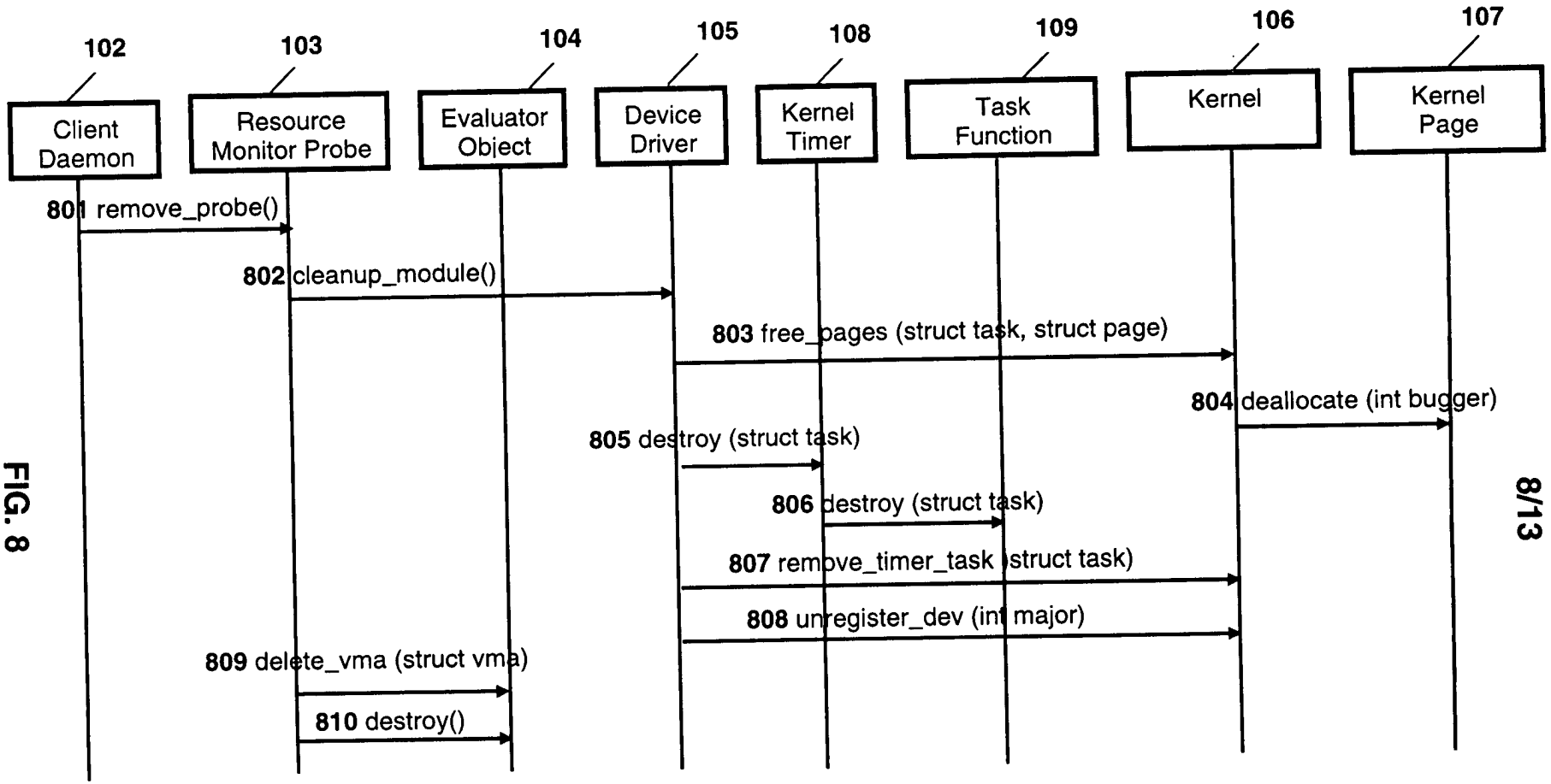
102
103
104
105
108
109
106
107
Client Daemon
Resource Monitor Probe
Evaluator Object
Device Driver
Kernel Timer
Task Function
Kernel
Kernel Page
801 remove_probe()
802 cleanup_module()
803 free_pages (struct task, struct page)
804 deallocate (int bugger)
805 destroy (struct task)
806 destroy (struct task)
807 remove_timer_task (struct task)
808 unregister_dev (int major)
809 delete_vma (struct vma)
810 destroy()

FIG. 8

```
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/mm.h>
#include <linux/malloc.h>
#include <linux/sched.h>

#define PROBE_MAJOR 0

struct file_operations cpu_probe_fops = {
        read:  probe_read,
        write: probe_write,
        open:  probe_open,
        release: probe_release,
        mmap: probe_mmap,
};

void *buffer;       // place to store the physical address of the kernel pages
void *head_buffer;  //current location to be written by our timer function
struct semaphore sem;

//use dynamic assignment
int probe_major =PROBE_MAJOR;
int order=0; //request for one page only

int init_module(void)
{
        result=register_chrdev(probe_major,"cpu_probe",&cpu_probe_fops);
        if(result<0) {
                printk(KERN_WARNING"cpu_probe: can't get major %d
\n",probe_major);
                return results;
        }
        if (probe_major==0) probe_major=result;

        buffer = (void*)__get_free_pages(GFP_KERNEL,order)
```

FIG. 9A

```
        if(!Buffer)
                goto nomem;

        memset(buffer,0,PAGE_SIZE<<order);
        return 0;

nomem:
        unregister_chrdev(probe_major,"cpu_probe");
        return 1;
}

void cleanup_module(void)
{
        if(buffer)
                free_pages((unsigned long)(buffer),order);

        unregister_chrdev(probe_major,"cpu_probe");

}

int probe_open(struct inode * inode, struct file *filp)
{
        MOD_INC_USE_COUNT;
        return 0;
}

int probe_release(struct inode *inode, struct file *filp)
{
        MOD_DEC_USE_COUNT;
        return 0;
}



int probe_read(struct file *filp, char * buf, size_t count, loff_t *fpos)
```

FIG. 9B

```
{
        if(copy_to_user(buf,(char*)buffer,PAGE_SIZE)))
        return -EFAULT;
        return PAGE_SIZE;
}

int probe_mmap(struct file *filp,struct vm_area_struct *vma)
{
        unsigned long offset=buffer;

        if(offset>=_ _pa(high_memory)||(filp->f_flags&O_SYNC))
                vma->vm_flags |=VM_IO;
        vma->vm_flags |= VM_RESERVED;
        If(remap_page_range(vma->vm_start,offset,vma->vm_end - vma->vm_start,
        return -EAGAIN;
return 0;
}

// pages[order] array of number of pages
/* the memory address returned by kmalloc and get_free_pages are also
virtual addresses.their actual value is still massaged by the MMU before
it is used to address physical memory */

wait_queue_head_t probe_wait;
struct timer_list probe_timer;

void probe_read_pid(unsigned long ptr)
{
        written =sprintf((char*)head_buffer,"%08u",(int)(current->pid));
        probe_data.len=+sizeof(int);
        probe_data.buf=buffer+sizeof(int);
        wake_up_interruptible(&probe_wait);
}
```

FIG. 9C

```
void probe_write(struct file *filp, char *buf, size_t count, loff_t *fpos)
{

        probe_data.len=0;
        probe_data.buf=buffer;
        probe_data.jiffies=jiffies;
        probe_data.queue=NULL;

        init_timer(&probe_timer);
        probe_timer.function = probe_read_pid;
        probe_timer.data=(unsigned long) &probe_data;
        probe_timer.expires=jiffies+HZ;       //one sec timeout
        add_timer(&probe_timer);
        interruptible_sleep_on(&probe_wait);
        del_timer_sync(&probe_timer);

        return 0;
}
```

FIG. 9D

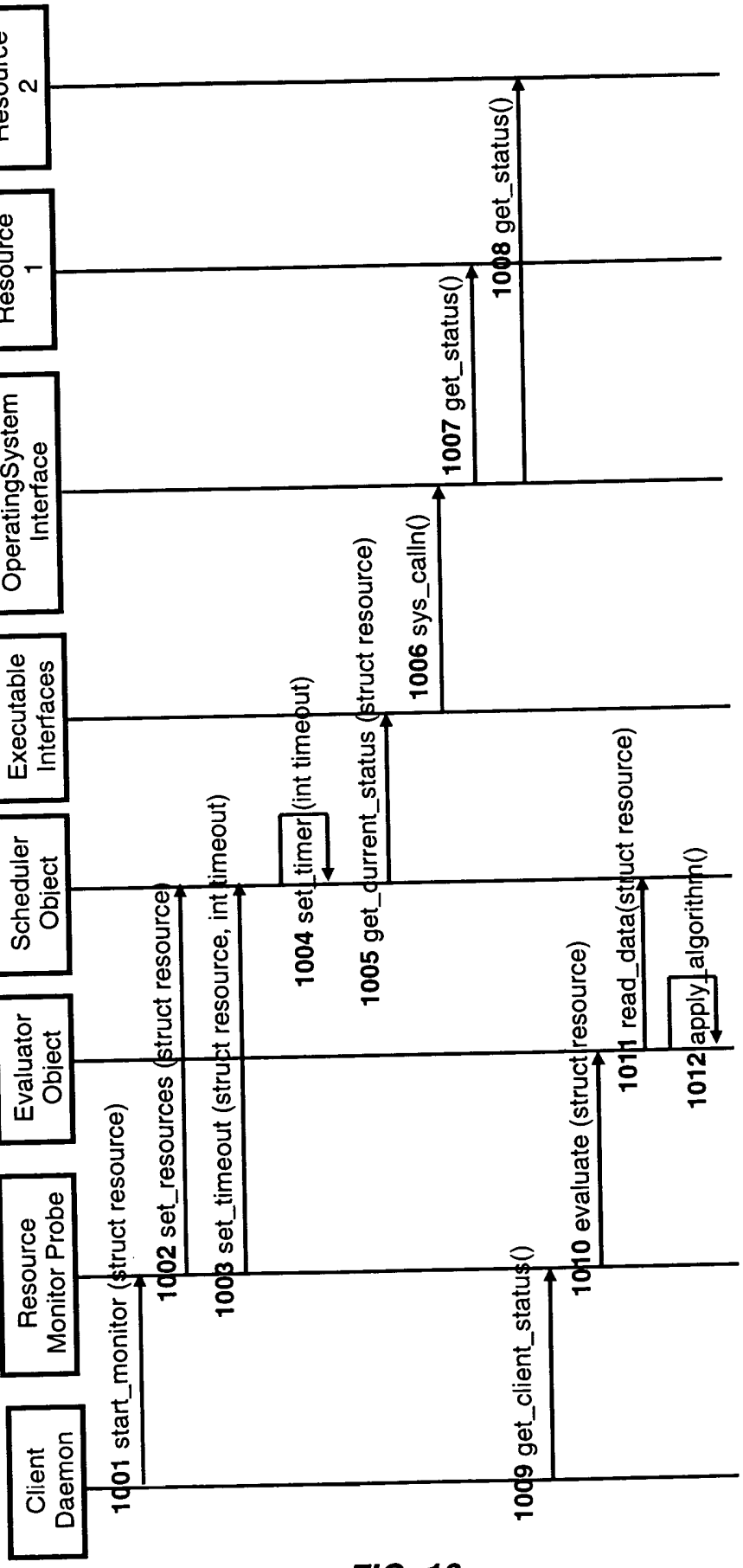
Client Daemon
Resource Monitor Probe
Evaluator Object
Scheduler Object
Executable Interfaces
OperatingSystem Interface
Resource 1
Resource 2
1001 start_monitor (struct resource)
1002 set_resources (struct resource)
1003 set_timeout (struct resource, int timeout)
1004 set_timer (int timeout)
1005 get_current_status (struct resource)
1006 sys_calln()
1007 get_status()
1008 get_status()
1009 get_client_status()
1010 evaluate (struct resource)
1011 read_data(struct resource)
1012 apply_algorithm()

FIG. 10